VILNIAUS GEDIMINO TECHNIKOS UNIVERSITETAS

FUNDAMENTINIŲ MOKSLŲ FAKULTETAS

INFORMACINIŲ TECHNOLOGIJŲ KATEDRA

**LABORATORINIS DARBAS 3**

Atliko: DGTfm 22 stud.

Matas Malickas

Priėmė: doc. dr. Andrej Bugajev

Vilnius, 2023

Darbo eiga

Užduoties numerio apskaičiavimas

Pagal formulę ((n − 1) mod 4) + 1, priskyrus n = 8, buvo apskaičiuotas užduoties skaičius: 4.

Užduotis

Ištirkite tiesinę priklausomybę parametro "BendrasLaikas" nuo parametro "svoris" (agreguodami pagal maršrutą ir datą, pritaikykite sumos operaciją), kai nagrinėjami duomenys tik su viena ta pačia reikšme "Masinos tipas". Analizės metu pritaikykite tiesinę regresiją.

Aplinkos paruošimas

Norint naudotis „Apache Spark“ bei kitomis reikalingomis bibliotekomis, lokalioje aplinkoje buvo įdiegti „pyspark“, „matplotlib“ bei „pandas“ pip paketai naudojantis komandomis:

pip install pyspark

pip install matplotlib

pip install pandas

Norint naudotis šių bibliotekų funkcijomis, buvo importuotos reikalmos funkcijos, tipai. Taip pat buvo inicializuota „Spark“ sesija bei užkrautas duomenų failas:

*import matplotlib.pyplot as plt*

*from pyspark.ml.feature import VectorAssembler*

*from pyspark.ml.regression import LinearRegression*

*from pyspark.sql import SparkSession*

*from pyspark.sql.functions import col, udf*

*from pyspark.sql.types import \**

*spark = SparkSession.builder.appName('Laboras3').getOrCreate()*

*text\_file = spark.sparkContext.textFile("duom\_cut.txt")*

Reikšmių išskyrimas iš failo ir mapinimas

Toliau yra apdorojamas ir nuskaitomas duomenų failas bei sudaromi raktai, kurie leis agreguoti duomenis pagal maršrutą bei sustojimo datą. Duomenys su neteisingomis reikšmėmis yra pažymimos ir paskui pašalinamos iš nuskaityto duomenų rinkinio.

def parsinam(line):

return line[2:len(line)-2].split('}}{{')

def parsinam2(line):

objs = line.split('}{')

k1 = None

k3 = None

k4 = None

k5 = None

for at in objs:

temp = at.split('=')

if (len(temp) < 2):

break

key, val = at.split('=')

if (key == 'marsrutas'):

k1 = val

if (key == 'sustojimo data'):

k3 = val

if (key == 'Masinos tipas'):

k4 = val

if (key == 'svoris'):

k5 = val

if (k1 != None and k3 != None and k4 != None and k5 != None):

return (k1+"\_"+k3, (k4, float(k5)))

else:

return (0, (1, "invalid"))

fmap = text\_file.flatMap(parsinam)

# Parse file and filter out invalid lines

mmap = fmap.map(parsinam2).filter(lambda x: x[0] != 0)

mmap.take(5)

# Aggregate byt key and sum values (svoris)

mmap = mmap.reduceByKey(lambda x, y: (x[0], x[1]+y[1]))

mmap.take(5)

Šioje būsenoje duomenys yra paruošti tolimesnėms operacijoms.

Maršrutų duomenų apdorojimas

Siekiant ištirti tiesinę priklausomybę parametro “BendrasLaikas” nuo parametron “svoris”, buvo apdoroti maršrutų duomenys bei taip pat suskirstyti į eilutes su raktais, kurie yra sudaryti iš maršruto ir sustojimo datos. Kadangi buvo atliekamas tyrimas su laiko tipo kintamuoju, laikas buvo paverstas į sveikąją reikšmę naudojantis vartotojo apibrėžta funkcija. Sudarius maršrutų RDD duomenų rinkinį, buvo išfiltruotos visos blogos reikšmės bei išmesti stulpeliai, kurie surado “ID” (rakto) reikšmę.

# Get unique 'Masinos tipas' values

unique = mmap.map(lambda x: x[1][0]).distinct()

unique.collect()

# Jusu darbas cia:

# Kodas, kito failo nuskaitymas ... duomenu agregavimas

routes = spark.read.option("header", True).csv("RouteSummary.txt")

routes.printSchema()

# Drop unused columns

routes = routes.drop("M", "BendrasAtstumas", "BendrasSvoris", "BendraKaina")

def makeID(str1, str2):

return str1+"\_"+str2

makeID\_UDF = udf(lambda z1, z2: makeID(z1, z2), StringType())

def convert\_time(string):

hours, minutes = string.split(':')

return int(hours) \* 60 + int(minutes)

convert\_time\_udf = udf(lambda z: convert\_time(z), IntegerType())

# Make ID and drop those columns

routes2 = routes.withColumn('ID', makeID\_UDF(

"marsrutas", "sustojimo data"))\

.drop("marsrutas", "sustojimo data")

routes2 = routes2.filter(col('BendrasLaikas').isNotNull())\

.withColumn('BendrasLaikas', convert\_time\_udf('BendrasLaikas'))

routes2.printSchema()

Priklausomybės analizė

Bendro laiko priklausomybei nuo svorio ištirti buvo atlikta duomenų analizė pasitelkiant „data frame“ struktūras. Priklausomybės analizei ištirti, buvo sudarytas ciklas, kurio reikšmės yra visos unikalios mašinos tipo vertės. Kievienam mašinos tipui buvo išfiltruoti duomenys (“ID” bei “svoris”) iš pateiktų duomenų. Šie duomenys buvo apjungti į vieną “data frame” su maršrutų duomenimis bei buvo pataisytas laiko tipas iš “string” į “integer”, kad būtų galima atlikti skaičiavimus. Naudojantis “pyspark.ml” programinės įrangos paketu, buvo sukurtas “VectorAssembler” objektas, kurio įvestis yra “svoris” reikšmės, o išvestis savybės (“features”).

Tam, kad duomenys būtų pateikti tiesinės regresijos modeliui, buvo sukurtas sąvybių vektorius bei iš jo pašalintos panaudotos bei nebereikalingos reikšmės. Iš savybių vektoriaus buvo sukurtas tiesinės regresijos modelis.

for u in unique.collect():

print(f'Value "Masinos tipas": {u}')

# Filter by 'Masinos tipas'

filtered\_data = mmap.filter(lambda x: x[1][0] == u)\

.map(lambda x: (x[0], x[1][1]))

# Convter RDD to DataFrame

data\_frame = filtered\_data.toDF(['ID', 'Svoris'])

joined\_data\_frame = data\_frame.join(routes2, 'ID')

joined\_data\_frame = joined\_data\_frame.withColumn(

'BendrasLaikas', joined\_data\_frame.BendrasLaikas.cast(IntegerType()))

# Create feature vector

vector\_assembler = VectorAssembler(

inputCols=['Svoris'], outputCol='features')

assembled\_vector = vector\_assembler.transform(joined\_data\_frame)\

.drop('Svoris').drop('ID')

linear\_regression = LinearRegression(

maxIter=10, regParam=0.3, elasticNetParam=0.8, featuresCol='features', labelCol='BendrasLaikas')

linear\_regression\_model = linear\_regression.fit(assembled\_vector)

Išsitreniravus modeliui (po metodo „fit“ iškvietimo) buvo galima sužinoti tiesinės regresijos lygties koeficientus, determinacijos koeficientą bei nubraižyti grafikus.

# Print the coefficients and intercept for linear regression

print("Coefficients: %s" % str(linear\_regression\_model.coefficients))

print("Intercept: %s" % str(linear\_regression\_model.intercept))

# Summarize the model over the training set and print out some metrics

trainingSummary = linear\_regression\_model.summary

print("numIterations: %d" % trainingSummary.totalIterations)

print("objectiveHistory: %s" % str(trainingSummary.objectiveHistory))

trainingSummary.residuals.show()

print("RMSE: %f" % trainingSummary.rootMeanSquaredError)

print("r2: %f" % trainingSummary.r2)

Kiekvienam mašinos tipo modeliui buvo nubraižyti tiesinės priklausomybės grafikai, kurie leidžia lengviau įžvelgti reikšmių pasiskirstymą bei suprasti ar koreliacija egzistuoja.

assembled\_vector.printSchema()

pandasDF = assembled\_vector.toPandas()

pandasDF.head()

labels = pandasDF['BendrasLaikas'].to\_list()

values = pandasDF['features'].to\_list()

print(labels)

print(values)

fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(20, 5))

axes.scatter(labels, values, s=10)

axes.set\_xlabel('BendrasLaikas')

axes.set\_ylabel('Svoris')

plt.show()